WIREGUARD
TAILSCALE
HEADSCALE

PIERRE GAMBAROTTO

Created: 2023-11-14 mar. 11:11

OBJECTIFS DE LA PRÉSENTATION

- wireguard : comprendre le principe
- réseau mesh : concept
- NAT : comment traverser
- tailscale/headscale : retour d’expérience

QUI SUIS-JE ?

- responsable info de l’IMT
- membre de la plmteam depuis 2021

Aime bien regarder des nouvelles technologies et en discuter après

INITIALEMENT

- assurer la sécurité physique des accès réseaux au laboratoire
- radius et 802.11x au niveau 2
- pourquoi pas un vpn, utilisable en externe et en interne ?

Recherche d’une solution plus légère qu’openvpn => wireguard

Gestion centralisée des clefs wireguard => tailscale

version opensource du serveur central tailscale => headscale

WIREGUARD

- connexion point à point sur udp
- vpn : au dessus d’une connexion existante
- chiffrement par paire de clef pub/priv
- instructions pour faire en manuel wiki Arch : wireguard

IMPLÉMENTATIONS

- noyau linux depuis fin 2019
- noyau windows depuis août 2021
- wireguard-go pour du multi OS
- boringtun en rust

=> supportés sur tous les OS

EXEMPLE : CONFIGURATION MANUELLE

Point à point entre A et B

- génération des clefs
- configuration de l’interface wireguard
- configuration des pairs

=> installer wireguard-tools, fournit :

- wg
- wg-quick

ÉTAPE 1 : LES CLEFS

```
# sur chaque pair A et B
wg genkey > /etc/wg/private_key
wg pubkey < /etc/wg/private_key > /etc/wg/public_key
```

Il faut copier la clef publique de A sur B, celle de B sur A

ÉTAPE 2

peer A, public IP 198.51.100.3

```
ip link add dev wg0 type wireguard
ip addr add 10.0.0.1/24 dev wg0
ip addr add fdc9:281f:04d7:9ee9::1/64 dev wg0
wg set wg0 private-key /etc/wg/private_key \
  listen-port 12345
wg set wg0 peer PEER_B_PUBLIC_KEY \
  allowed-ips 10.0.0.2/32,fdc9:281f:04d7:9ee9::2/128
ip link set up dev wg0
```

A écoute sur 195.51.100.3:12345

peer B has no static IP

```
ip link add dev wg0 type wireguard
ip addr add 10.0.0.2/24 dev wg0
ip addr add fdc9:281f:04d7:9ee9::2/64 dev wg0
# no listen : will be dynamic
wg set wg0 private-key /etc/wg/private_key
wg set wg0 peer PEER_A_PUBLIC_KEY allowed-ips 10.0.0.2/32,fdc9:281
  endpoint 198.51.100.3:12345
ip link set up dev wg0
```

B n’écoute initialement pas pour des connexions, il peut seulement en initier une.

VÉRIFIER LA CONF

```
wg show wg0
```

peer B

```
interface: wg0
  public key: UhmH1vJBPmV2cG6w2gC5WNX7k35gutV+tdfe3rCaRRs=
  private key: (hidden)
  listening port: 38461

peer: TjR1IfrK6L7JUeoFJ+rDjQ+Oveq3XAFO/mH90I7ZKQc=
  endpoint: 195.51.100.3:12345
  allowed ips: 10.100.0.1/32
  latest handshake: 17 minutes, 48 seconds ago
  transfer: 348 B received, 3.66 KiB sent
```

peer A

```
interface: wg0
  public key: TjR1IfrK6L7JUeoFJ+rDjQ+Oveq3XAFO/mH90I7ZKQc=
  private key: (hidden)
  listening port: 41641

peer: UhmH1vJBPmV2cG6w2gC5WNX7k35gutV+tdfe3rCaRRs=
  endpoint: 130.120.38.57:38461
  allowed ips: 10.100.0.2/32
  latest handshake: 19 minutes, 39 seconds ago
  transfer: 404 B received, 3.23 KiB sent
```

endpoint créé quand B s’est connecté à A !

OUTILS POUR AUTOMATISER

- wg-quick, permet de gérer des fichiers de conf

```
[Interface]
  Address = 10.200.100.8/24
  DNS = 10.200.100.1
  PrivateKey = oK56DE9Ue9zK76rAc8pBl6opph+1v36lm7cXXsQKrQM=

  [Peer]
  PublicKey = GtL7fZc/bLnqZldpVofMCD6hDjrK28SsdLxevJ+qtKU=
  PresharedKey = /UwcSPg38hW/D9Y3tcS1FOV0K1wuURMbS0sesJEP5a
  AllowedIPs = 0.0.0.0/0
  Endpoint = demo.wireguard.com:51820
```

- systemd-networkd
- …

BILAN WIREGUARD

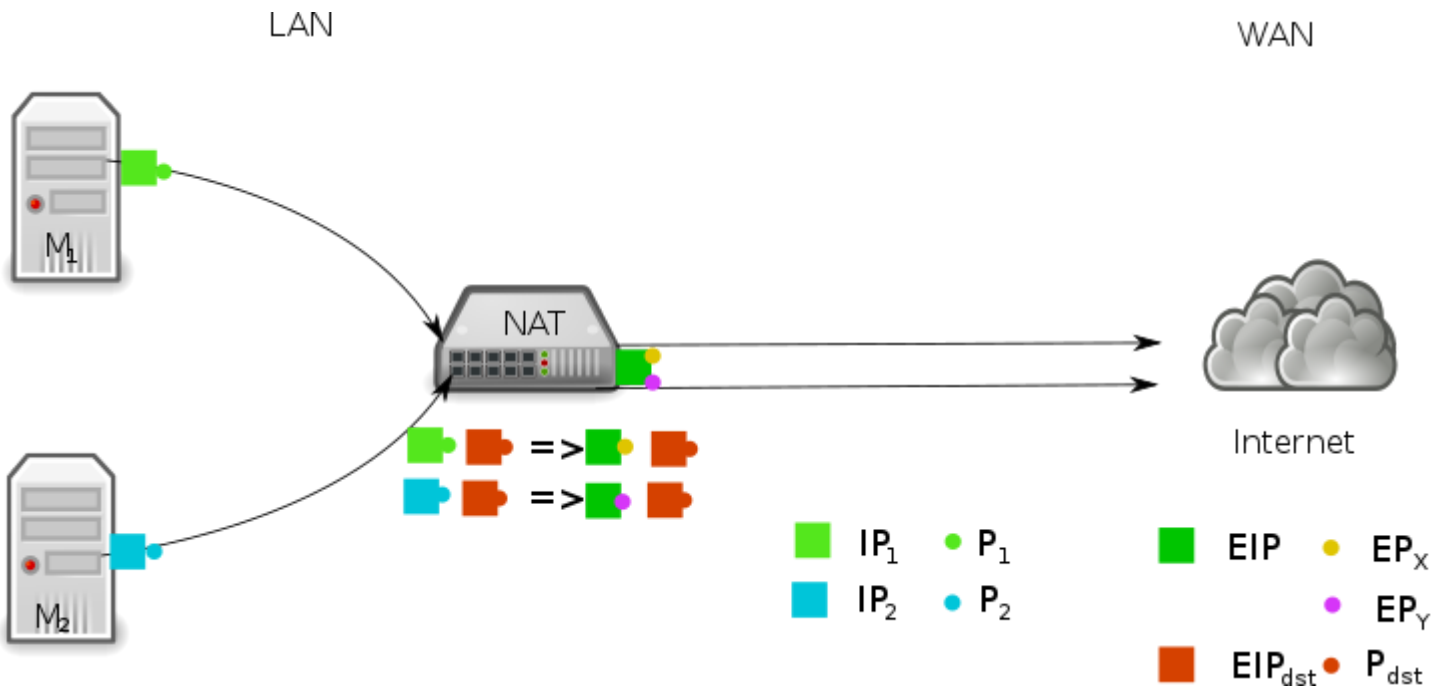
simple pour du point à point, performant, stable

vite compliqué :

- gestion des infos des pairs (clef/endpoint/ip ranges)
- NAT pour les IP privées
- NAT traversal : à faire
- gestion DNS : à faire

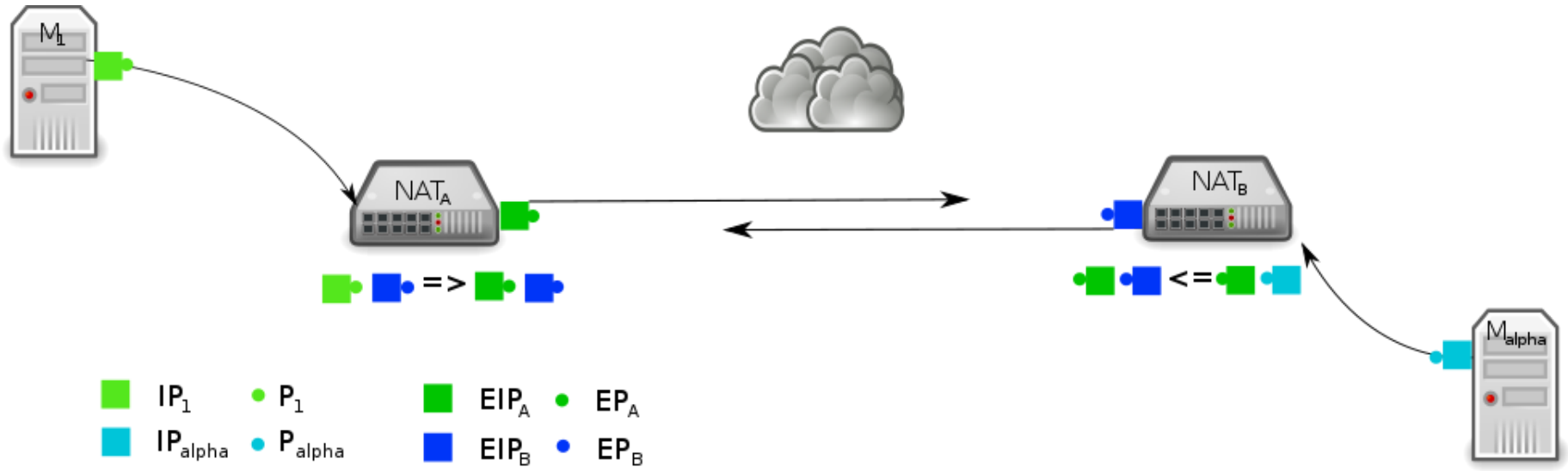
NAT

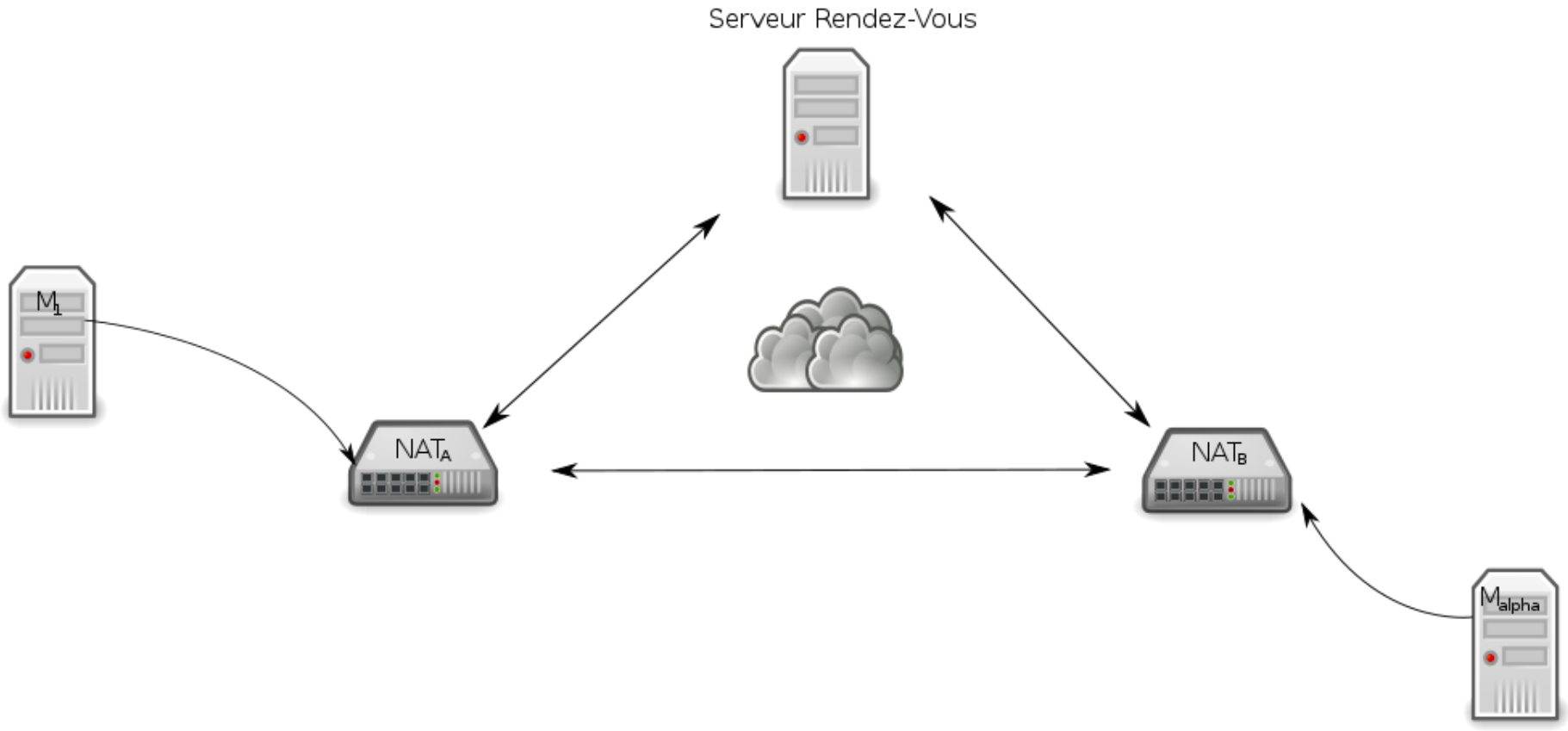
(Source) Network Address Translation



UDP HOLE PUNCHING

Comment faire communiquer 2 machines qui n’ont pas d’IP publiques ?





chaque machine doit connaître son (IPext, PExt) pour la communiquer

STUN/ICE : on passe par un serveur tiers

NATS

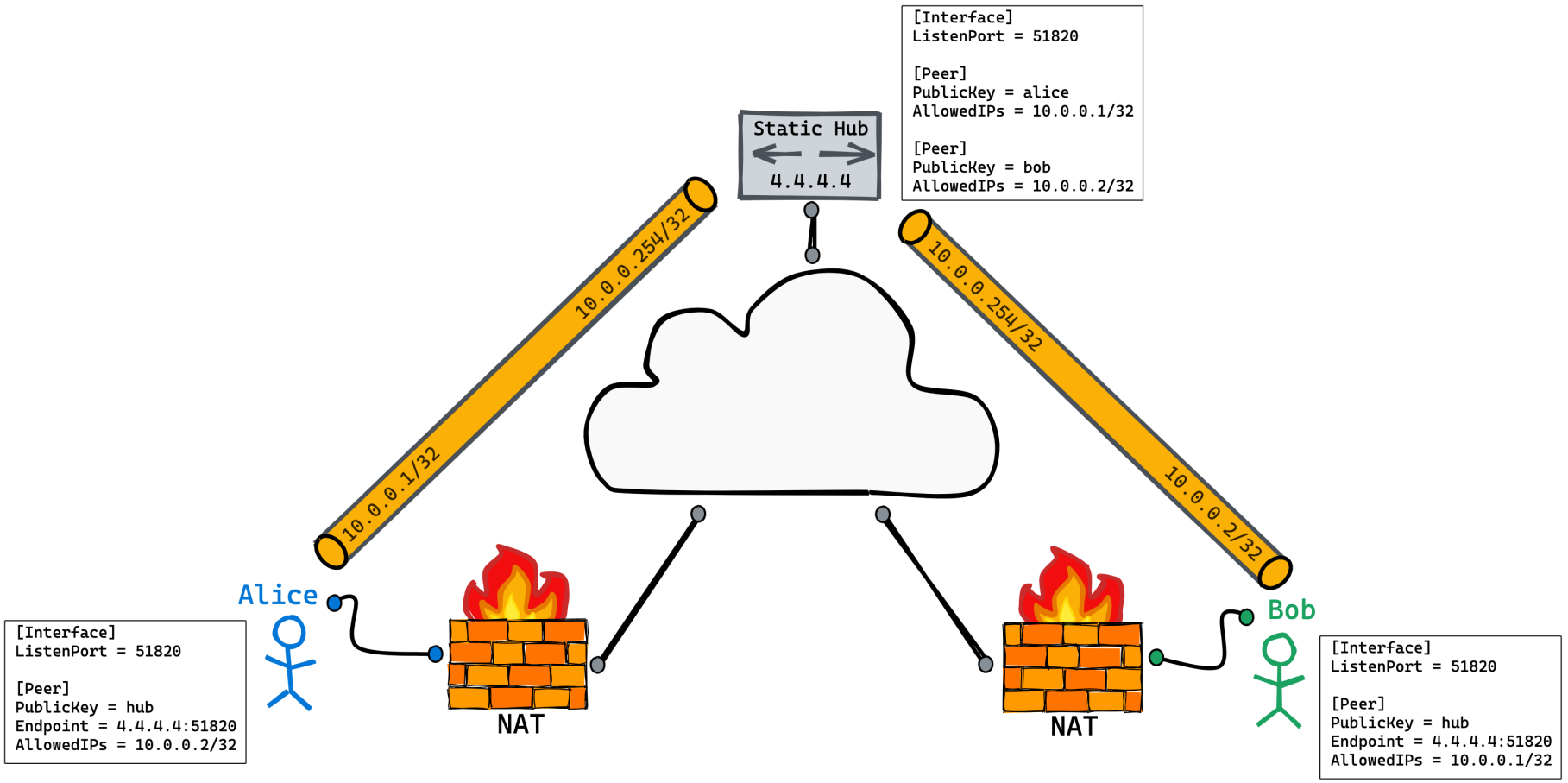
Différents types de NAT

Au pire : tunnel par un hôte tiers : TURN server

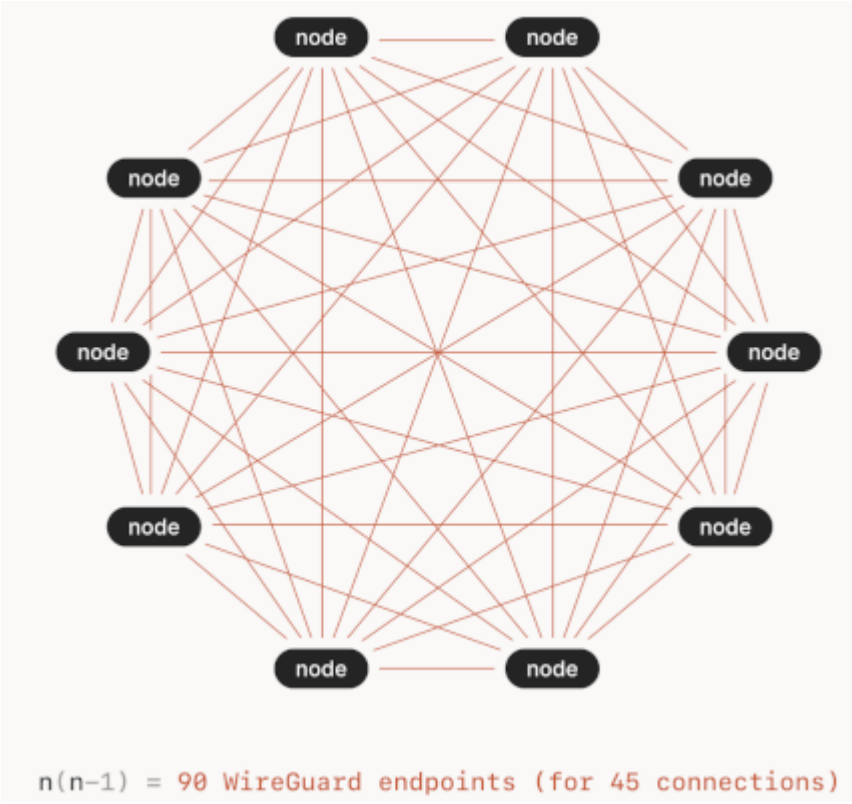
Tailscale et Nat Traversal : gère tous les cas

AU DELÀ DU POINT À POINT

Besoin d’un serveur STUN, pourquoi ne pas faire :



RÉSEAU MESH


node
node
node
node
node
node
node
node
node
node
n(n−1) = 90 WireGuard endpoints (for 45 connections)

MESH SUR WIREGUARD

copier les infos des pairs sur tous les nœuds

idée :

- centraliser la gestion des clefs
- garder la connexion directe par wireguard

MESH + NAT

en résumé :

- hub & spoke pour la gestion des pairs
- hub & spoke pour STUN, voire TURN en mode dégradé
- point to point pour la connexion entre les pairs

=> étude des solutions basées sur ce modèle

https://github.com/cedrickchee/awesome-wireguard

FONCTIONNALITÉS

Feature\Software	Open source	Free	Full Mesh	Auto conf	Devices	Supports Users	Allows full tunnel	Subnet Access	NAT traversal	Linux	Windows	MacOS	Android	IOS	OpenWRT	Custom DNS
Vanilla WireGuard	✅	✅	❌	❌	Unlimited	❌	✅	✅	❌	✅	✅	✅	✅	✅	✅	✅
Tailscale	✅ ! [0]	❌ [free]	✅	✅	Unlimited 100	✅ 3	✅	✅	✅	🌐	🌐	🌐	🌐	🌐	✅	✅ ! [3]
Headscale	✅	✅	✅	✅	Unlimited	❌	✅	✅	✅	🌐	🌐	🌐	🌐 ! [2]	🌐 ! [2]	✅	✅
Netmaker	✅ ! [1]	✅	✅	✅	Unlimited	✅	✅	✅	✅	🌐	🌐	🌐	✅	✅	✅	✅
WGSD	✅	✅	✅	❌	Unlimited	❌	✅	❌	✅	✅	❌	❌	❌	❌	❌	❌
Innernet	✅	✅	✅	❌	Unlimited	✅	✅	❌	✅	✅	❌	✅	❌	❌	❌	
Wesher	✅	✅	✅	✅	Unlimited	❌				✅	❌	❌	❌	❌	❌	❌
Netbird	✅	✅	✅	✅	Unlimited 20	✅	✅	✅	✅	✅	✅	✅	❌	❌	✅	✅
wgmesh	✅	✅	✅	✅	Unlimited	❌	✅	❌	❌	🌐	❌	❌	❌	❌	❌	❌
wiresmith	✅	✅	✅	✅	Unlimited	❌	❌	❌	❌	🌐	❌	❌	❌	❌	❌	❌

source : github:MarvsG/WireGuardMeshes

CE QUI M’INTÉRESSE

- NAT Traversal
- clients pour les OS majeurs, les mobiles sont un plus
- DNS : ajout des machines du réseau vpn

CHOIX : TAILSCALE/HEADSCALE

Pourquoi tailscale :

- NAT traversal qui marche
- architecture propre
  - serveur de coordination : gestion des clefs, STUN
  - serveurs DERP : équivalent TURN spécialisé pour wireguard

headscale : implémentation opensource du serveur de coordination tailscale

TAILSCALE

- clients multi os opensource et packagés
- serveur en mode SAAS fermé et commercial
- soutient le projets opensource headscale

RÔLE DU SERVEUR DE COORDINATION

- stocker les clefs publiques
- stocker les endpoint (comme STUN)

Principe global :

client tailcale –> connection wireguard sur le serveur de configuration :

- obtient son statut NAT -> IP:Port public ou bascule sur DERP
- renseigne ses infos de connexions (publicKey, allowedIps, name)
- récupère les infos des autres pairs

FIREWALL

What firewall ports should I open to use tailscale ?

- device -> control server/derp : inside -> tcp *:443
- device -> device : udp:41641 -> **:**
- device -> udp *:3478 stun to derp servers
- tcp:80, tcp:443, icmp in and out, udp:3478 : derp server

HEADSCALE

https://headscale.net/running-headscale-linux/

- implémentation opensource du serveur de contrôle de tailscale
- en go
- fait par rétro ingénierie du client tailscale
- reconnu et soutenu par tailscale

limite de base : un seul réseau mesh de supporté

DÉPLOIEMENT PAR NIXOS

frontal nginx

```
{ config, lib, pkgs, ... }:

let
    proxy-hostname = "ts.math.univ-toulouse.fr";
    proxy-uri = "https://${proxy-hostname}";
    backend-uri = "http://130.120.38.54:8080";
in
{
  services.nginx.virtualHosts."${proxy-hostname}" =  {
    enableACME = true;
    forceSSL = true;

    locations."/" = {
      proxyPass = "${backend-uri}";
      proxyWebsockets = true; # needed if you need to use WebSocke
    };
  };
}
```

backend :

```
{ config, lib, pkgs, flakes, ... }:
{
  environment.systemPackages = [ config.services.headscale.package
  networking.firewall = {
    allowedTCPPorts = [ 8080 ];
  };
  services.headscale = {
    enable = true;
    address = "0.0.0.0";
    port = 8080;
    settings = {
      ip_prefixes = [ "100.64.0.0/10" ];

      server_url = "https://ts.math.univ-toulouse.fr";
      dns_config = {
        override_local_dns = true;
        nameservers = [ "172.22.1.7" ];
        base_domain = "math.univ-toulouse.fr";
        domains = [ "servers.math.univ-toulouse.fr" "math.univ-tou
        magic_dns = true;
      };
```

```
    };
}
```

UTILISATION

- connecter un client

```
tailscale up --login-server https://ts.math.univ-toulouse.fr
```

- lister tous les pairs contactables :

#+begin_src conf tailscale status 100.64.0.1 alba gamba linux - 100.64.0.3 headscale.servers.math.univ-toulouse.fr servers linux active; direct 130.120.38.54:41641, tx 20056 rx 18392 100.64.0.2 in.servers.math.univ-toulouse.fr servers linux offline 100.64.0.5 oneplus-in2023 gamba android offline 100.64.0.4

tsrouter.servers.math.univ-toulouse.fr servers linux active; direct 130.120.38.57:41641, tx 517348 rx 2733380 ¶#+end$_{src}$

AJOUTER UNE MACHINE AVEC UNE CLEF PRÉ GÉNÉRÉE SUR LE SERVEUR DE CONTRÔLE

clef permanente créé côté serveur headscale :

```
headscale --user bob preauthkeys create --reusable --tags tag:serv
# expiration possible
```

On peut la trouver avec

```
headscale preauthkeys list --user bob
```

Sur une machine cliente à connecter :

```
tailscule up --login-server https://ts.math.univ-toulouse.fr --aut
```

EXIT-NODE/SUBNET-ROUTING

Pour ne pas avoir à installer tailscale sur toutes les machines existantes

- exit-node : nœud du réseau mesh pour route par défaut
- subnet-routing : idem mais en spécifiant les routes

EXIT-NODE

sur le nœud de sortie :

```
# activate routing : echo 1 > /proc/sys/net/ipv4/ip_forward
tailscale set --advertise-exit-node
```

sur headscale :

```
headscale routes list
# le serveur est dispo pour fournir une route par défaut
heasdcale routes enable -r 3
```

sur un autre nœud

```
tailscale set --exit-node headscale
```

ROUTAGE SUBNET

tailscale: subnets

```
# sur le nœud faisant le routage
tailscale set --advertise-exit-node=false --advertise-routes \
  "130.120.36.0/22,130.120.80.0/22,\
   130.120.224.0/22,\
   172.22.0.0/16"

# sur le serveur headscale, activer les routes
headscale routes list
headscale routes enable -r X # à faire pour chaque route

# sur le client :
tailscale up --force-reauth \
  --login-server=https://ts.math.univ-toulouse.fr \
  --accept-routes
```

tailscale utilise une table de routage séparée :

```
ip route show table 52
100.64.0.2 dev tailscale0
100.64.0.3 dev tailscale0
100.100.100.100 dev tailscale0
130.120.36.0/22 dev tailscale0
130.120.80.0/22 dev tailscale0
130.120.224.0/22 dev tailscale0
172.22.0.0/16 dev tailscale0

# ip rule # to show priority
# ip route show table all
```

MON EXPÉRIENCE JUSQUE LÀ

- client facile à installer
- infra facile à déployer
- un utilisateur = une clef headscale, permet d’enregistrer plusieurs nœuds
- linux et android ok, le reste à tester
- subnet-routing : permet de remplacer openvpn rapidement

Encore à tester :

- gestion des ACLs
- déployer son propre serveur DERP

Évolution : à attendre ou à coder soi-même

- automatiser la création des clefs utilisateurs après authentification
- déduire les ACLs de l’authentification

LIMITES

Gestion actuelle des clefs : génération sur le serveur headscale

tailscale : auth oidc => clef d’enregistrement pour un utilisateur

Cela commence à arriver côté headscale, mais pas encore au point sur la gestion des *claims*. => échec pour le moment avec l’openid plm

acls : un seul fichier texte pas dynamique

BILAN

- bonne solution pour remplacer un vpn «admin»
- de l’enrobage à faire pour réaliser un vpn «utilisateurs»
- évaluer DERP pour passer outre les DSI trop frileuses sur les ouvertures UDP